FINAL 01 JUN 92 TO 31 MAY 95

THE ACQUISTION AND UTILIZATION OF SPATIAL AND FUNCTIONAL KNOWLEDGE FOR IMAGERY ANALYSIS

F49620-92-J-0318
2304/GS 61102F

DAVID M. KCKEOWN JR.

CARNEGIE MELLON UNIVERSITY
DIGITAL MAPPING LABORATORY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH, PA 15213

AFOSR-TR-96
0101

AFOSR/NM
110 DUNCAN AVE, SUITE B115
BOLLING AFB DC 20332-0001

F49620-92-J-0318

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

13. ABSTRACT (Maximum 200 words)

SEE REPORT FOR ABSTRACT

19960320 066

14. SUBJECT TERMS			15. NUMBER OF PAGES
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFED	20. LIMITATION OF ABSTRACT SAR

The Acquisition and Utilization of Spatial and Functional Knowledge for Imagery Analysis .

Final Technical Report
F49620-92-J-0318
June 1st, 1994 through May 31st, 1995

David M. McKeown, Jr.
Principal Investigator
Digital Mapping Laboratory
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA. 15213

Dr. Abraham Waksman
COTR
Mathematical and Information Sciences
Air Force Office of Scientific Research
Bolling AFB, Building 410
Washington, D.C. 20332

1 Executive Summary

In June 1992, researchers at the Digital Mapping Laboratory, School of Computer Science at Carnegie Mellon University, began work on a 36 month contract to explore the acquisition and utilization of spatial and functional knowledge for imagery analysis as supported under AFOSR Contract F49620-92-J-0318[1]. Over the course of this contract, we have built upon our previous research in large-scale knowledge-based systems for the interpretation of aerial imagery. This previous work has focused on knowledge acquisition, analysis and evaluation of system performance, and scaling large knowledge-based computer vision systems to work on more complex image datasets. Under this contract we continued work in these areas, particularly in scaling of the system and began the port of the SPAM system to a distributed memory multicomputer architecture that utilizes an ATM switch for low latency high bandwidth communication. During final year of the contract, we have extended our previous work in parallelizing large rule-based systems for distributed memory architectures. Additionally, we continued our efforts in developing better language support for parallel production systems. This final technical report briefly reviews past progress as well as serves to describe the research results during the third year of our research contract.

1.1 Accomplishments

The third and final year of our program of research June 1994 until June 1995 have been focused on modifications to the MIDWAY runtime system to allow for improved parallel execution of our knowledge based system SPAMThe following is a brief summary of the contents of this report.

We have continued our collaboration with the MIDWAY group at CMU and have extended our previous work in task-level parallelism. Further experiments using the SPAM/TLP system implemented on the MIDWAY distributed shared memory system have resulted in improved parallel performance. Our current results, run on a network of 8 DEC 5000/200's, show improved speed-ups (5.9-fold using 7 child processes, up from previous results of 4.5-fold). These improvements are mostly a result of a detailed analysis of the SPAM/TLP system on MIDWAY. In addition to improving performance, we moved toward a complete SPAM/MIDWAY system by implementing the third phase of SPAM (FA) under MIDWAY. On the MIDWAY side, we have been working to improve the robustness of the MIDWAY system. Specifically, we have

[1]This work was sponsored by the Air Force Office of Scientific Research under Contract F49620-92-J-0318. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Air Force Office of Scientific Research, the Advanced Research Projects Agency, or the United States Government.

removed dependencies on a MACH network server. This server has been a source of problems for our experiments from the beginning and, with our latest research further stressing MIDWAY, these problems had been exacerbated. We have also invested a great deal of effort in instrumentation, performance analysis, and tuning of MIDWAY to support the type of high contention lock accessing behavior that the FA phase of SPAM presented to MIDWAYIn addition, an ATM interface was successfully integrated into the MACH operating system kernel. This now gives SPAM/MIDWAY researchers the flexibility to easily select the underlying type of network communication desired (either ATM or Ethernet). Further details on this work can be found in Section 3. Finally, in Section 4, we discuss several areas of future work, conclusions, and lessons learned.

1.2 Publications

The following technical reports, conference papers, and journal articles describe research results that were produced under the support of this research contract.

1. Brustoloni, J. C. (1994). "Exposed Buffering and Sub-Datagram Flow Control for ATM LANs" in *Proceedings of the 19th Annual Conference on Local Computer Networks*, November, 1994.

2. McKeown, D., et al., (1994). "Research in the Automated Analysis of Remotely Sensed Imagery: 1993–1994" in *Proceedings of the DARPA Image Understanding Workshop*, November, 1994, pp 99–132.

3. Zekauskas, M., Sawdon, W, and Bershad, B. (1994). "Software Write Detection for a Distributed Shared Memory" in *Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI)*, November, 1994.

1.3 Researchers Supported

The following faculty, staff, and graduate students were fully or partially supported under research contract F49620-92-J-0318:

David M. McKeown, Jr. is a Principal Research Computer Scientist in the School of Computer Science at Carnegie Mellon University and has been a member of the research faculty since 1986. He received a B.S. degree in Physics and an M.S. degree in Computer Science from Union College, Schenectady, N.Y. Prior to joining the faculty he was a researcher at Carnegie Mellon from 1975, a Research Associate at George Washington University and a member of the Technical Staff at NASA Goddard Space Flight Center, Greenbelt Maryland (1974-1975), and an Instructor in Computer Science and Electrical Engineering at Union College (1972-1974).

His research interests in computer science are in the areas of image understanding for the analysis of remotely sensed imagery, digital mapping and image/map database systems, computer graphics, and artificial intelligence. He is the author of over 35 papers and technical reports and is an active consultant for government and industry in these areas.

He has been the principal investigator on research programs sponsored by the U.S. Army Topographic Engineering Center, the Defense Mapping Agency, the Air Force Office of Scientific Research, and the Advanced Research Projects Agency. He is a member of ACM, IEEE, AAAI, American Society for Photogrammetry and Remote Sensing, and Sigma Xi. He is Co-Chair of the International Society for Photogrammetry and Remote Sensing (ISPRS) Intercommission Working Group II/III on Digital Photogrammetric Systems (1992-1996) and serves on the Editorial Board of the Journal of Intelligent Information Systems, Kluewer Academic.

Wilson Harvey is a Project Scientist in the School of Computer Science at Carnegie Mellon University. He received a B.S. in Physics and Mathematics from Carnegie Mellon University in 1986. Mr. Harvey has been with the School of Computer Science for more than six years, working primarily on the development of knowledge-based computer vision systems and the representation of spatial knowledge for scene analysis. His research interests include knowledge representation and image understanding. He is a member of Sigma Xi.

Dirk Kalp is a Senior Research Programmer in the School of Computer Science at Carnegie Mellon University. He graduated from Carnegie Mellon University in 1973 with a B.S. degree in Mathematics and from the University of Pittsburgh in 1981 with an M.S. degree in Computer Science. On the research staff at Carnegie Mellon since 1985, he has worked primarily in the area of parallel production systems. In addition to parallel architectures his research interests include multiprocessor operating systems, virtual memory systems, and the development of tools for fitting symbolic parameter cognitive models.

2 Background

We begin this section with a description of SPAM, our scene interpretation system. Then, as a context for the research described in the remainder of this report, we describe our relevant work in the areas of parallelism and language support for scaling large knowledge-based systems.

2.1 The SPAM Architecture

SPAM is a production system architecture for the interpretation of aerial imagery with applications to automated cartography and digital mapping [McKeown *et al.*, 1985; McKeown *et al.*, 1989]. It tests the hypothesis that the interpretation of aerial imagery requires substantial knowledge about the scene under consideration. Knowledge about the type of scene, whether airport, suburban housing development, or urban city, aids in low-level and intermediate level image analysis, and will drive high-level interpretation by constraining search for plausible consistent scene models. SPAM has been applied in two task areas: airport and suburban house scene analysis.

As with many computer vision systems, SPAM attempts to interpret the 2-dimensional image of a 3-dimensional scene. The particular goal of the SPAM system is to interpret an image segmentation, composed of image regions, as a collection of real-world objects. For example, the output, given an input image containing an airport, would be a model of the airport scene, describing where the runway, taxiways, terminal-building(s), etc., are individually located. SPAM uses four basic types of scene interpretation primitives: *regions*, *fragments*, *functional-areas*, and *models*. SPAM performs scene interpretation by transforming image *regions* into scene *fragment* interpretations. It then aggregates these fragments into consistent and compatible collections called *functional-areas*. Finally, it selects sets of functional-areas to form *models* of the scene.

The first phase of SPAM is called region-to-fragment (RTF). This is a traditional heuristic classification process, using knowledge about the classes of features that occur in the scene to map a segmentation to a set of interpretations. Local properties of the segmentation, such as shape, texture, or height, are used to select which interpretations to generate. Examples of the type of knowledge used in region-to-fragment would be *runways are typically 50 to 80 meters wide*, or *houses are 8 to 10 meters high.*

The second phase of SPAM is called local-consistency check (LCC). This phase performs a modified constraint satisfaction between the interpretations generated in region-to-fragment. The knowledge in this phase consists of the constraints between the different pairs of objects. An example constraint would be *runways have perpendicular taxiways.* There are many such constraints in the system — some numeric (e.g., distance) and some non-numeric (e.g., intersection). Each constraint provides weak support to the participating hypotheses. Thus, it is not the action of a single constraint, but the collective action of several constraints which causes two interpretations to be found consistent with one another.

The functional-area (FA) phase groups together those interpretations that support one another, where support is computed from the results of the previous phase (LCC). A functional-area is defined as a group of interpretations that are similar in function and are in close physical proximity to one another. For instance, a terminal functional-area is defined to contain only terminal-building, parking-lot, road, and parking-apron interpretations. It is physically represented by the convex hull of the features in the functional-area.

Finally, the model-generation (MODEL) phase uses the functional-areas and combines them based on a number of heuristics, including number of conflicts, number of supported interpretations, and area of coverage. Conflicts are identified and resolved, either using support within the context of the model, or by invoking some process that would provide additional knowledge. For example, if in the context of a model a region of the image was interpreted as both a taxiway and a hangar-building, a stereo process could be invoked to help resolve the conflict based on the region's height estimate. Multiple models are commonly generated and these models can be used as contexts for further processing.

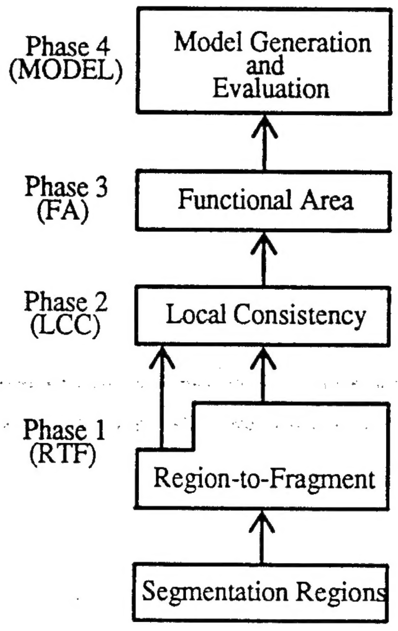


Figure 1: Interpretation phases in SPAM.

As shown in Figure 1, each interpretation phase is executed in the order given. SPAM drives from a local, low-level set of interpretations to a more global, high-level, scene interpretation. There is a set of hard-wired productions for each phase that control the order of rule executions, the processing of geometric computations, and other domain-independent tasks. However, this "bottom-up" organization does not preclude interactions between phases. For example, prediction of a fragment interpretation in the *functional-area* (FA) phase will automatically cause SPAM to reenter the *local-consistency check* (LCC) phase for that fragment. Other forms of top-down activity include stereo verification to disambiguate conflicting hypotheses in the *model-generation* (MODEL) phase and to perform linear alignment in the *region-to-fragment* (RTF) phase.

Another way to view the flow of processing in SPAM is that knowledge is used to check for consistency among hypotheses; contexts are created based on collections of consistent hypotheses, and are then used to predict missing components. A collection of hypotheses must combine to create a context from which a prediction can be made. These contexts are refinements or spatial aggregations in the scene. For example, a collection of mutually consistent runways and taxiways might combine to generate a runway functional area. The context of a runway functional area then predicts that certain sub-areas within that functional area are good candidates in which grassy areas or tarmac regions may be found. However, an isolated runway or taxiway hypothesis cannot directly make these predictions. In SPAM the context determines the prediction. This serves to decrease the combinatorics of hypothesis generation and to allow the system to focus on those areas with strong support at each level of the interpretation.

2.1.1 Related Work

There has been a large body of work in traditional knowledge-based systems to support knowledge acquisition and validation [Gupta, 1991]. Typical issues include the validation of knowledge-based systems, checking for completeness and consistency in knowledge bases, and system performance evaluation. The application areas are broad, spanning medical diagnosis, design, process control, and configuration analysis. However, little work in this area has been focused within the context of computer vision tasks. One explanation is that there are few end-to-end knowledge-based systems for computer vision in the literature. A corollary is that those knowledge-based systems that have been built are 'one-person' systems that do not survive much past the thesis defense. Some of the earliest 'high-level' vision systems applied to aerial image analysis include Matsuyama's system [Matsuyama, 1980], the Sigma system [Hwang, 1984], and SPAM[McKeown *et al.*, 1985]. More recently, researchers have explored the use of generic knowledge [Huertas *et al.*, 1989] and investigated the formalization of knowledge to support high-level vision [Strat and Smith, 1988].

Our previous work in this area [McKeown *et al.*, 1989] focused on the use of compilation tools to translate a high-level schema based constraint representation into OPS5 productions. We also developed static analysis tools to aid in the display and debugging of the model descriptions generated by SPAM. As a result of the long term use of this system, our research focus has shifted toward issues in automating portions of the knowledge acquisition process and in the development of tools to aid in the diagnostic analysis of finer levels of system behavior. After a description of some of our previous work, we will discuss our recent work in these areas.

2.1.2 Knowledge Acquisition

Previous work has focused on the use of compilation tools, such as our RULEGEN compiler, to translate a high-level schema based constraint representation into OPS5 productions [McKeown *et al.*, 1989]. We also developed static and interactive analysis tools (spats, rtfchk, lccchk) to aid in the display and debugging of the intermediate and final model descriptions generated by SPAM. In order to do a detailed analysis of the system's performance, tools were built and detailed ground-truth data was compiled for six different airport scenes and six suburban housing scenes. As a result of the long term use of these tools, our research focus shifted toward issues in automating portions of the knowledge acquisition process and in the development of tools to aid in the diagnostic analysis of finer levels of system behavior.

In extending this work, the usefulness of the interactive knowledge acquisition tools inspired us to build an interactive browsing and performance analysis tool (spamevaluate). This helped to improve the user's ability to refine SPAM's knowledge base, as well as to broaden the methods available to the user for performance analysis. The complexity of the SPAM system, coupled with the large data sets sizes, prompted us to look at methods for automatic knowledge acquisition. Several automated tools were built (rtfanalyze, lccanalyze, faanalyze) to examine the ground truth data and construct rules based on simple statistical measures.

Recently, better understandings of system behavior, additions to the knowledge representation, and the addition of new tools have prompted further work in this area. Refinement of this suite of tools continues as the SPAM architecture evolves.

2.2 Scaling Large Knowledge-Based Systems

As SPAM's knowledge base grew, so did the end-to-end running time. It became apparent that a moderate increase in the size of SPAM's knowledge base would result in significant increases in the running time; so much so that it would impair our ability to do research. Typical SPAM runs took between 10 to 100 hours on a VAX 11/780. It also seemed clear, from the results of performance analysis, that more knowledge would be necessary to improve SPAM's interpretation results. In order to address the need for more processing efficiency, we began joint research with members of the Production System Machine group at CMU to explore the utility of task-level parallelism in the context of SPAM. This work produced several interesting results, including a new SPAM system built on CParaOPS5, a C-based implementation of OPS5, and extensive experience using shared memory multi-processors to achieve near linear speedups.

During the final year (1991–1992) of our previous AFOSR contract, AFOSR-89-0199, we were able to capitalize on our ability to perform larger scale experiments using the CParaOPS5 environment and extend our research on task-level parallelism to include multiple multi-processors communicating using a network shared memory mechanism. With the advent of 100+ MIPS uniprocessor workstations like the DEC Alpha, and the inability of large shared memory machines to keep pace, this research effort concentrated on the efficiency of our implementation of OPS5. However, workable distributed shared memory programming environments like PVM [Geist *et al.*, 1993] and MIDWAY have prompted us to revive our investigation of task-level parallelism but using networked high performance workstations in place of multi-processor architectures.

Our previous work in parallel architectures developed the concept of task-level parallelism. Task-level parallelism is obtained by decomposing the production system at a high level, thereby exploiting the inherent parallelism in the task. Previously, we investigated the use of task-level parallelism (TLP) [Harvey *et al.*, 1989; Harvey *et al.*, 1990] for speeding up large-scale production systems. SPAM was used as the vehicle for this research performed in conjunction with the Production System Machine group at Carnegie Mellon University. Using task-level parallelism we achieved almost linear speed-ups running on a 16-processor Encore Multimax. In addition, this work produced CParaOPS5 [Acharya and Kalp, 1990], an optimized, C-based version of ParaOPS5 [Kalp *et al.*, 1988] (an assembly language compiler for parallelizing OPS5 programs).

With the availability of high performance uniprocessor workstations now exceeding 100 MIPS, it seemed prudent to suspend our parallelism research until the performance of stable multiprocessor platforms again surpassed uniprocessors. Recently, the MACH group at Carnegie Mellon finished work on an initial implementation of a parallel programming environment called MIDWAY. This environment allows a user to program distributed memory multiprocessors as if they were a shared memory multiprocessor. This is similar to our previous work with the virtual shared memory system [Forin *et al.*, 1989; Li, 1988; Harvey *et al.*, 1990]. In our case, the distributed shared-memory multiprocessor is a set of workstations with a high-speed ATM network interconnect. The MIDWAY system allows us to exploit the inherent parallelism in SPAM while continuing to utilize the fastest uniprocessor workstations as they become available.

2.2.1 MIDWAY

MIDWAY is a system for writing and running shared memory parallel programs on distributed memory multiprocessors [Bershad and Zekauskas, 1991; Bershad *et al.*, 1993]. It combines the programmability of a shared memory multiprocessor with the scalability of a distributed memory system. MIDWAY programs are written in conventional programming languages, such as C and C++. Concurrency within a program is expressed using *threads*, and controlled using *locks* and *barriers*. For example, a MIDWAY program could be written and debugged on a Sequent Symmetry, a true shared memory multiprocessor, and then compiled and relinked for execution on a Delta Touchstone distributed memory multiprocessor.

Figure 2 shows the hardware architecture used for our experiments with Midway. It consists of eight DEC 5000/200 workstations[2] (individually rated at approximately 25 MIPS), each with 64Mb of memory and running MACH3.0. The machines are networked using both 10 Mbit Ethernet and 140 Mbit ATM interfaces. The ATM hardware consists of eight Fore Systems TCA-100 ATM adapter cards connected via a Fore ASX-100 ATM switch. The MIDWAY system supports both network interfaces; the communication method used is selectable in software.

MIDWAY provides the programmer with a new model of memory consistency called *entry consistency*. Entry consistency takes advantage of the relationship between specific synchronization variables which protect critical sections and the shared data accessed within those critical sections. In an entry consistent system, a processor's view of memory becomes consistent only when it enters a critical section. The only shared memory that is guaranteed to become consistent is that which can be accessed within the critical section. This model provides no greater consistency than that required by most shared memory parallel programs. Consequently, it has the potential for a higher performance implementation than a memory model which delivers to a program "more" consistency than necessary.

MIDWAY consists of three main components: a set of simple extensions to a base language (in this case, C) in the form of keywords and function calls that are used to annotate a parallel program, a runtime system that implements the entry consistency model, and a pre-compiler that translates annotated MIDWAY programs into the underlying base language. The pre-compiler is written in C (actually, extensions have been added to the GNU C compiler). The runtime library provides a portable, machine-independent interface to entry consistent shared memory across a wide range of processor and interconnect architectures. The initial MIDWAY implementation runs on a network of MIPS-based workstations connected

[2]Faster machines, such as the DEC Alpha workstation, rated at approximately 100 MIPS, are available and are compatible with our communication hardware. Using this hardware would require spending time to port MIDWAY to such a platform. We have chosen not to pursue such a port at this time because our research is focused on parallelism and utilization, and not on absolute system performance.

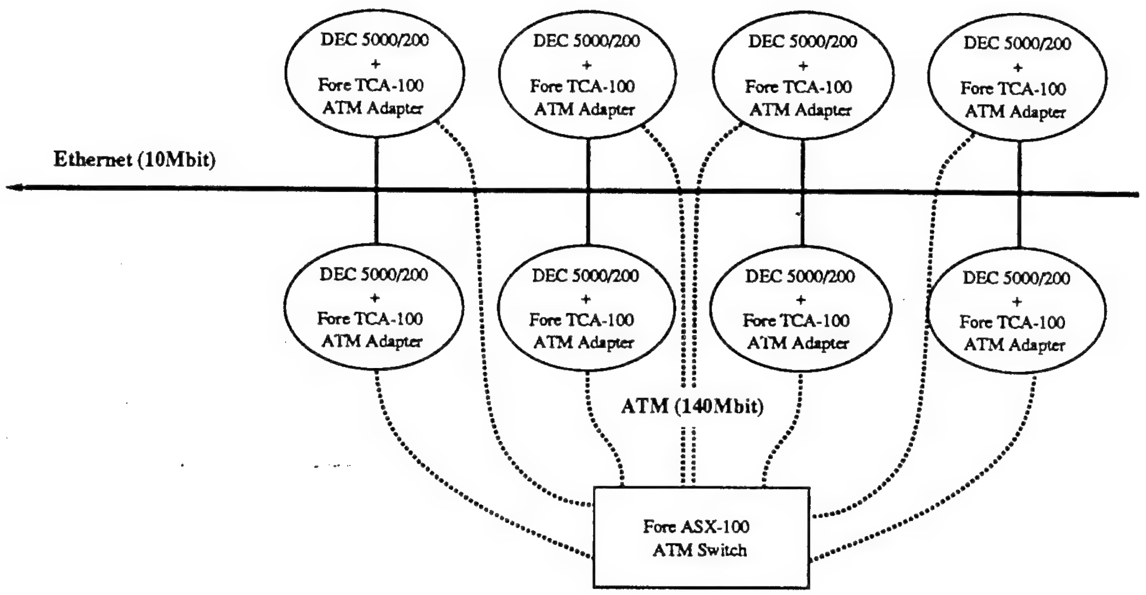


Figure 2: Hardware configuration for the MIDWAY distributed shared-memory experiments.

by ethernet or ATM/fiber links. Subsequent ports will handle other specialized distributed memory multiprocessors.

3 Task-Level Parallelism Under MIDWAY

The SPAM image interpretation system is a large knowledge-based system with over 700 OPS5 productions and taking more than four CPU-hours to run. Some of our previous work addressed SPAM's runtime performance, making 10–20 fold improvements by using better hardware and software technologies, *e.g.*, converting from Lisp to C, using highly optimized production system languages like CParaOPS5 [Acharya and Kalp, 1990; Kalp *et al.*, 1988], and running on high-performance workstations. However, these sources of speed-up are insufficient to keep pace with the modifications we believe are necessary to expand SPAM's task domain.

We then began to focus on the use of task-level parallelism (TLP) for speeding up large-scale production systems like SPAM [Harvey *et al.*, 1989; Harvey *et al.*, 1990]. Task-level parallelism is obtained by decomposing the production system at a high level, thereby exploiting the inherent parallelism in the task. We achieved near linear speed-ups on a shared-memory multiprocessor. Continuing our investigation of task-level parallelism, our most recent efforts have used virtual shared memory on a set of workstations with a high-speed network interconnect [McKeown *et al.*, 1994]. The MIDWAY system, a shared-memory abstraction built by the MACH group at CMU for use on distributed memory multiprocessors, has been the vehicle for this work.

Figure 3 shows a diagram of a generic shared memory architecture where the application program is distributed across a number of processors and associated memory. In our case, the interconnection can be either an Ethernet or an ATM switch. The Midway system provides a software layer that abstracts the fact that memory is associated with particular processors and allows the application to view nearly all memory in a homogeneous fashion. The abstraction has a set of costs and requires sophisticated memory management and consistency mechanisms to allow for maintaining coherency during the execution of the distributed application. In this section, we describe our efforts towards using the MIDWAY mechanisms in a complex knowledge-based vision system. This work has raised significant issues not generally explored when evaluating parallelism using traditional scientific applications.

During the second year of this research contract, we focused our efforts on integration and performance issues. We have parallelized SPAM's third phase (FA) and integrated it with the previously parallelized second phase (LCC). Extensive analysis of both the LCC and FA systems has uncovered several inefficiencies within MIDWAY and SPAM's use of MIDWAY.

In this third year of the research, we have had a very tight focus on the interaction between SPAM and MIDWAY. We have had to do extensive instrumentation and analysis on both systems to address performance issues and problems that SPAM's FA phase revealed in the MIDWAY design.

3.1 Parallelizing SPAM's Third Phase: FA

Toward achieving our goal of building an end-to-end SPAM system, we have built a parallel version of SPAM's third phase (FA) on MIDWAY. The FA phase groups objects together based on how well they support one another. The same model used to parallelize LCC was used to parallelize FA:

- Identify tasks (groups of productions) that can be executed in parallel.
- Modify the initialization productions so that a queue of tasks is created.
- Identify and remove any side effects in right-hand side (RHS) actions for parallel execution.

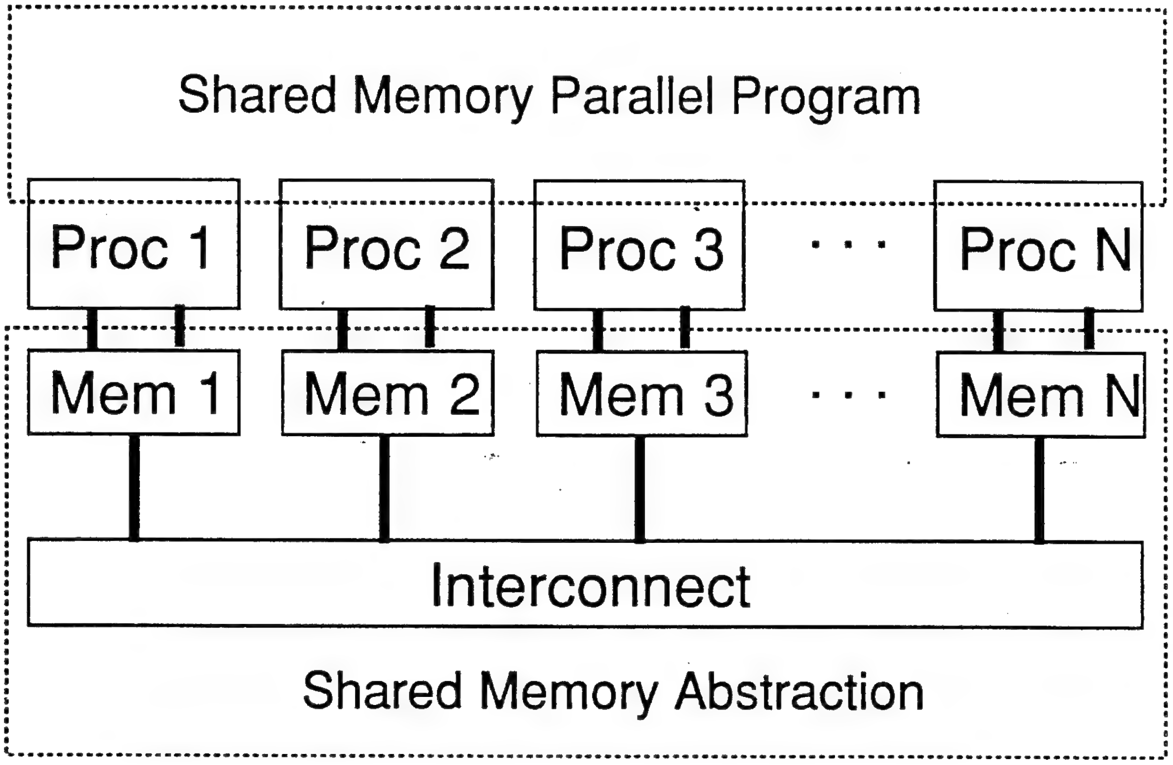


Figure 3: Diagram of distributed shared-memory.

Though the FA phase was never previously parallelized, its implementation using the TLP model was straightforward both because our previous experience had provided insight and because the FA phase fit the task-queue model well.

The next task was to integrate this phase with the previous one. Several changes had to be made to the SPAM/MIDWAY system so that the two phases could be connected together:

1. Data, in the form of working memory elements, returned from the slaves to the master has to be unbound from slave data locks and re-bound to a master data lock. In this way, the master (parent) processor recovers the data produced by the slave (child) processors during LCC, post-processes it, and redistributes it to the slaves at the beginning of the FA phase.

This feature, called *dynamic binding*, is missing from MIDWAY, but will be added in the near future. An interim solution is to have the master copy the LCC results and distribute the copy to the slave processes.

2. One particular type of data structure, the Rete hash table, must be shared between master and slaves and has a special type of consistency requirement. The hash table contains the internal state of the OPS5 Rete match network. The master generates this while initializing the task queue for each SPAM phase. A slave needs the master's Rete state in order to proceed with task processing, during which time a slave produces updates to the Rete state. However, the individual slave updates to this state are relevant only to the particular tasks a slave processes and do not need to be reflected back to the master or other slaves. Thus, these updates can effectively be discarded at the end of each phase. Furthermore, these Rete state changes are too fine-grained to be communicated among master and slaves to maintain a globally consistent monolithic Rete state across all parties. The cost would be prohibitively high.

   What the slaves need here is a particular type of sharing which supplies them with the current copy of the master's Rete state at the beginning of each phase — in essence, a memory snap-shot of the data structure. Snap-shot memory is a violation of MIDWAY's model of memory consistency, so it is unlikely to be made available in the MIDWAY runtime library. We have implemented a work-around to this by restricting slaves to read-only sharing of the master's Rete state and then having each slave create its own local copy at the start of each SPAM phase.

A schematic diagram of the entire system is shown in Figure 4. Once these modifications were completed, the combined system was ready to be tested.

Once our initial implementation was debugged, we began our performance tests by running a parent process with a single child process on a small data set. The hardware for our tests consisted of two DEC 5000/200's, each with 64 megabytes of memory running MACH 3.0, and networked together using a Fore Systems ATM switch. Of the 150.5 seconds of total execution time, 55% of that (82.9 seconds) was spent in task processing and 37% (55.6 seconds) was spent in writing the results to a file. The remaining time was spent in initialization and moving results between child and parent machines. The write-results time is a significant portion of the total time, yet our understanding of this portion of the system suggests that this computation should be very efficient. The data involved, about 3.5 megabytes stored in shared-memory, is read sequentially and processed before being written to the disk. This is performed by the parent after all the result data has been transferred back from the children. Thus, no network activity is required to fetch the shared data. Furthermore, the shared data is not modified during this operation and thus does not generate any bookkeeping overheads

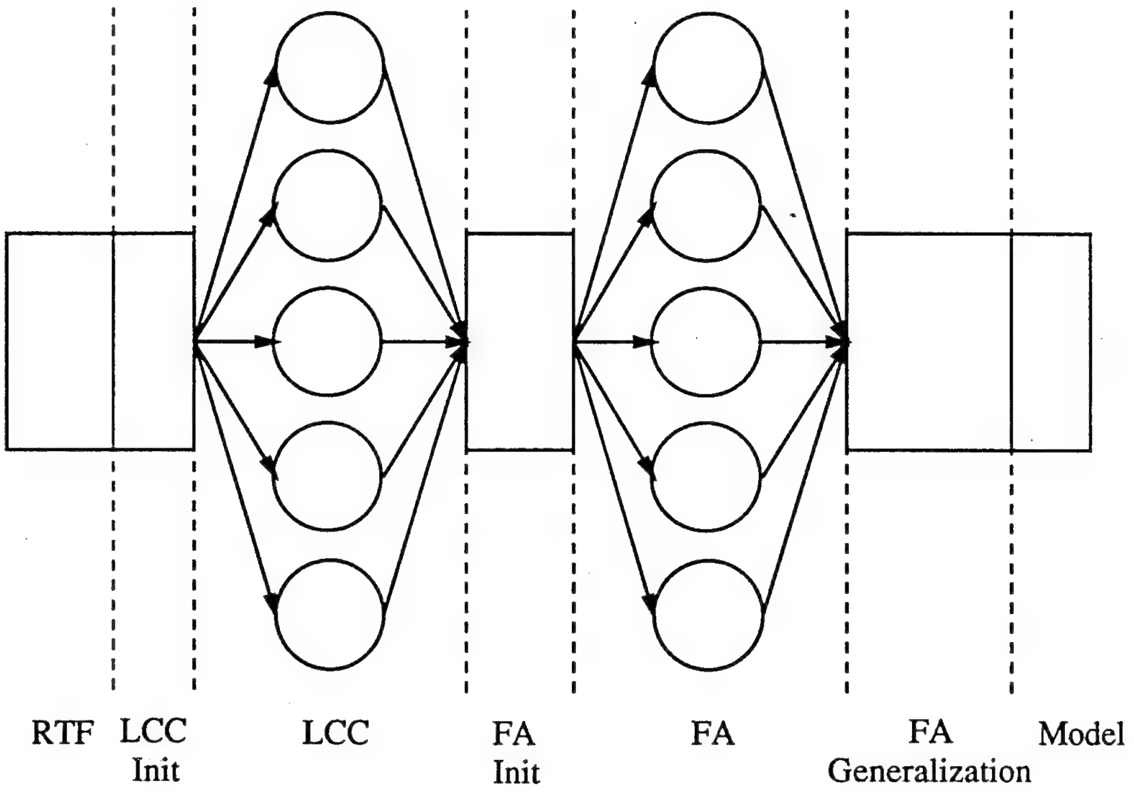


Figure 4: Execution flow of the SPAM/MIDWAY system.

for MIDWAY in dirty-bit processing. The MIDWAY overheads should be low and the write-results processing time should be comparable to the time for processing the same amount of data on a non-MIDWAY version of the system (about 28 seconds).

From a previous analysis of LCC, we were aware of inefficiencies in the write-results portion of the system. This experiment reaffirmed it as a bottleneck and revealed that a more detailed analysis of the SPAM/MIDWAY system would be necessary to understand and fix the problem.

3.2 SPAM/MIDWAY System Analysis

Implementing the FA phase gave us further insight into some of the performance issues we encountered when working with the LCC phase, It also served to expose additional performance and implementation issues in SPAM and MIDWAY. In our analysis of both the LCC and FA phases, we have broken the execution down into the following computation segments:

- Parent start-up
- Task-queue initialization
- Thread creation
- Child initialization
- Task-processing
- Return-results
- Write-results

Most often, dividing the processing into well-defined functional parts has helped us to isolate and eliminate the individual problems. We have instrumented the SPAM/MIDWAY system to keep timing and page-fault data for each of these segments. In conjunction with these measurements, we used the execution-profiling tools on our system to obtain details on the amount of time and number of calls to critical functions. We added a memory management layer to SPAM that is in direct correspondence with MIDWAY's underlying memory allocation mechanism and memory organization. We instrumented this to track dynamic allocations according to both size and data type and report these statistics at points in the processing. This allowed us to monitor the usage of shared memory in each of the segments and served to guide our investigation into performance overheads. The MIDWAY internal performance monitors were used, and we instrumented versions of the kernel so that we could examine kernel state. Finally, a low priority, but always run-able, idle thread was added to MIDWAY to allow us to use the profiler to report its execution time and thus indicate how long the application was blocked by a system call.

This instrumentation helped us to track and solve several problems, and it produced several more. In particular, we obtained fairly unreliable timing data when measuring the execution time across several system runs for each computation segment. To help solve this, we built a stand-alone version of the MIDWAY runtime system that would allow us to boot MACH in single-user mode and isolate our network branch from the system-wide network. We wanted to make sure that no daemon processes or external events could consume any network bandwidth or other system resources except those related to the application's execution.

With the stand-alone system working and the instrumentation in place, we found that MIDWAY's dirty bit processing code [Zekauskas *et al.*, 1994] (which tags shared data which has been written) was slowing down the child initialization and task-processing segments, and so it was further optimized. We also found that a large portion of time was spent in lock management, mostly due to a very large number of repeated lock acquires done in write-results. Optimizing the lock acquire path when the lock is locally held resolved this issue.

With the division of SPAM into computation segments, our timing measurements revealed that significant time was being spent in task-queue initialization and in task-processing. We broadly characterize the problems we encountered as:

- Problems during task-queue initialization
  - Instrumentation (lack of monitoring tools)
  - Lock management
  - Data-structure allocation/optimization
- Problems during task-processing
  - Instrumentation
  - Dynamic binding of locks to data
  - Bugs in MACH communication servers
  - Lock management and communication
  - Lock contention

As can be seen, the performance issues in SPAM/MIDWAY have tended to be a combination of things rather than a single issue. This has made it more difficult to identify causes and formulate solutions. In the following sections, we describe our efforts to fix the problems outlined above.

3.2.1 Problems During Task-Queue Initialization

The overheads in the initialization segments for both LCC and FA included a large component due to large volumes of shared data being shipped to the slave processes. The master processor generates this data as part of the initialization to set up the processing for the slaves. The shared data bound to locks provides the basis for MIDWAY's entry consistency protocol for cache management. Each lock acquire by a process causes the updates to the data bound to that lock to be shipped over the network to the process. In LCC and FA, the slave processors must acquire a variety of locks in order to obtain all the required data from the master's initialization. In response to this performance issue, the MIDWAY developers added address coalescing to maintain lock associated data in sorted order and consolidate consecutive address ranges. This would minimize the amount of lock to address mapping information shipped around the network and make shared data transfers more efficient by reducing the message count.

When this impact was not realized for the task-queue initialization segment of FA (in fact its performance worsened), we had to take a closer look at SPAM. MIDWAY organizes shared memory as a set of 8 megabyte regions. Each 8 megabyte region has an associated cache line size which is a power of 2. Shared allocations are made from the region of the appropriate cache line size. The association of shared data to a lock is then maintained by an address mapping table which identifies which cache lines in the regions are bound to the lock. Obviously, keeping the table sorted and coalescing address ranges reduces the size of the table and the overhead in servicing a lock acquire. The performance improvement gained from using the address coalescing code was expected to be significant for SPAM since it tends to have large numbers of one data type associated with a given lock, and each data type is dominated by a single size. Thus, most of the data bound by each lock would come from just one of the 8 megabyte cache line regions, providing good opportunity for consolidation. What we found from looking at SPAM's memory management statistics and later confirmed by some additional monitoring on the FA task-queue initialization segment on the master, was that one of SPAM's less significant data types — an internal table of dynamically generated strings — was the culprit. It was being allocated over a variety of cache line sizes and at regular intervals over the FA task-queue initialization period with just enough frequency to disrupt the address coalescing mechanism in MIDWAY, forcing its sorting algorithm down its worst-cost path almost every time. To address this problem, we set a minimum 64 byte fixed size on allocations that would serve for the bulk of the string table elements. It is interesting to note that some of the efforts of the past at optimizing memory efficiency in both SPAM and CParaOPS5 have had to be undone in the new context of distributed shared memory. For performance, we must trade memory for network latency.

3.2.2 Problems During Task-Processing

Having resolved some of the above issues, we were ready to focus on the parallel performance of FA in its task processing segment. The performance here was a serious issue as the task processing segment actually slowed down instead of speeding up when processors were added. Here the FA parallelization uncovered some deficiencies in MIDWAY and pointed to the need for some major revisions and improvements to MIDWAY in several areas. The parallel implementation of the FA phase provided the first opportunity on MIDWAY to investigate finer-grain symbolic computation under a distributed shared memory system. Unlike previous scientific benchmarks implemented on MIDWAY, SPAM and, in particular, the FA phase provided a "real" application example of non-scientific code that stressed MIDWAY in different ways. In particular, the shared data structures are very dynamic and the communication demands very high. A revised approach to dynamic binding of locks and data was required. The higher communication demands uncovered two problem areas: unresolvable bugs in the MACHdependent communications server and performance issues and bugs in the MIDWAY locking and communication code.

Addressing the problem of dynamic binding in SPAM required a complete analysis of the dynamic data structures in SPAM and their lock binding characteristics. Dynamic re-binding changes the association between synchronization and the data that synchronization protects. The FA phase of SPAM uses data structures that grow and shrink dynamically, illustrating the need for efficient dynamic re-binding. We collected address traces on the dynamic data bindings according to data type. The traces were used to drive a simulator we built that modeled the MIDWAY address binding mechanism and permitted new algorithms to be tested. With this, we studied methods to more efficiently implement dynamic re-binding. The results of this analysis were then used to provide the basis for a redesign of dynamic binding in MIDWAY. In lieu of its implementation, we resorted to an interim solution involving memory preallocation in address ranges specific to each data type so we could proceed in the exploration of the other issues involving communication demands.

To address the first problem related to the increased communication demands, we re-implemented services to remove dependencies on the Mach netmsgserver which provided the basis for 3 important communication services in MIDWAY: the status message server, Unix IO server, and remote process invocation. We re-implemented all three of these using Unix sockets in order to remove the dependencies upon Mach. This had the added benefit of improving the portability of MIDWAY to non-Mach based platforms and provide the opportunity for future comparison to other distributed shared memory systems.

The second problem of performance issues and bugs in MIDWAY's locking and communications code required an extensive amount of work to instrument MIDWAY and SPAM and analyze detailed traces of the locking operations. The trace logs were augmented with timestamps on each entry derived from a free running hardware counter driven at the processor clock rate (40 ns on a 5000/200). The performance issues raised by FA had not been previously encountered by the other applications ported to MIDWAY. These applications were primarily scientific problems where the data structures and algorithms have a very regular character and easy decomposition. This provides the opportunity to assign a set of locks to a data structure, such as the rows of a matrix, and the contention for the data structure is reduced since a process need only acquire the lock associated with a small part, such as a single row of the matrix. In SPAM, data structures like the symbol table and the dynamic string table do not have such regular decompositions or predictable access patterns that would admit any attempted decomposition. The result is that the lock for the data structure can become a bottleneck when contention is high. In the FA phase of SPAM, the contention for the lock is high because the tasks are fine grained, 2 orders of magnitude smaller than the LCC tasks, and thus computation time between lock acquires is also very small. FA accesses 3 such locks during its processing. The high demands placed on MIDWAY by FA thus uncovered performance issues in the design of the midway locking and communication layers not previously encountered by other applications. It also exercised untested parts of the code and revealed a number of difficult bugs that we had to track down and fix along the way in investigating performance.

There were a variety of different problems and issues to resolve most of which were related to contention. MIDWAY relied on a simple mechanism to forward a lock acquire request to the current lock owner, queuing the request there until the lock was free. But the full support for the queuing mechanism was not implemented. When the lock became free, the lock was transferred to the first process on the queue but the rest of the queue was not shipped along. This actually worked fine in the low contention cases such as the scientific applications as well as SPAM's LCC phase because there was usually at most only 1 process ever waiting in the queue. Because network communication is not always reliable, there was also a MIDWAY daemon thread on each processor that was responsible for retransmitting a request that was not serviced before a timeout period. This retransmission mechanism provided the backup for those few instances in the scientific applications where the queue might have more than one process waiting. The added overhead for the few retransmissions did not impact those applications in any significant way. But for FA, our investigations showed that the queue often had over half the processes waiting, thus retransmissions were a performance issue. Besides the negative impact of retransmission activity, performance was also being degraded by network activity in forwarding requests from processor to processor to find the current lock owner. MIDWAY relies on just local knowledge at each processor about who that processor knows the last lock owner to be, i.e., its best guess. Besides degrading performance, both the retransmission and request hopping activity presented some race conditions not encountered before that stressed the communication layer in MIDWAY uncovering bugs in the transaction table mechanism that manages network message traffic. Messages sent from a host on the network may arrive at the other host in opposite order. With the high contention in FA, this created problems for us in lock request messages as well as lock invalidate messages.

We resolved all these problems and implemented the full queuing mechanisms, but the net effect yielded only about a 20% improvement in FA task processing – still not enough to produce any speedups. Further instrumentation revealed that the operation to transfer the lock is a problem for one particular lock, the dynamic string table lock. Lock transfer involves 2 components: marshalling the data, bound to the lock, that's changed into a message and then sending the message. Both of these components suffer a performance problem, taking a factor of 2-5 times longer than the symbol table lock, for example. The data marshalling component has the most serious problem and may require a redesign of the MIDWAY dirty bit mechanism and algorithms used to tag the lock associated data that's changed. This may require a significant amount of work thus relegating it as a topic for future work for MIDWAY. An alternative here would be to try to redesign the SPAM dynamic string table to accommodate some type of decomposition to associate a set of locks. But with a large SPAM library sitting on top of this data structure, the effort required here is also extensive.

3.3 Modifications to MIDWAY

For SPAM, MIDWAY serves two avenues of research. First it permits us to build upon our previous investigations of using task-level parallelism in SPAM and demonstrating the effectiveness of distributed shared memory to exploit it. With the expected success of these efforts, it then provides us with the opportunity to leverage off that work and use MIDWAY as a viable tool to enhance the scope and capability of the vision research in SPAM. Thus, we anticipate MIDWAY to survive beyond its own particular research agenda and goals and become a stable substrate upon which we can build some of our future SPAM vision research. This raises the issue of support for maintenance and future development of MIDWAY. Our solution to this is to provide for it from within the SPAM group. To this end, we have invested effort in eliminating the use of MACH's netmsgserver in MIDWAY.

For MIDWAY, the netmsgserver serves as a conduit for network communication related to startup of MIDWAY application processes on remote host machines, regular Unix file I/O, and the logging of MIDWAY status messages. The netmsgserver has proven to be the source of a number of bugs in its interaction with MIDWAY and is suspect in others. In particular, for the SPAM/MIDWAY work, problems with intermittent hanging and crashing were traced to the netmsgserver. Such instabilities made it difficult to perform experiments, particularly when the number of processors increases. In addition, our stand-alone system had the unfortunate effect of exacerbating this instability problem. Thus removing the netmsgserver and replacing it with a Unix socket-based design will make MIDWAY more robust and enhance its portability by eliminating a MACH dependency.

We have now eliminated the use of the netmsgserver from MIDWAY, having designed and implemented a MIDWAY status message server, Unix I/O server, and remote process invocation, all based on Unix sockets.

Another major effort was the merging of a "standard" MACH kernel (supporting Ethernet) with an experimental kernel which supported ATM-based network communication. The unified kernel now gives us the flexibility to easily select either underlying communication network to support distributed shared memory. Although the higher-bandwidth ATM interface is preferred, not all our machines have ATM interfaces. Because Ethernet is a standard component, more machines can be utilized in experiments. In addition, performance measurements on the Ethernet-based system continue to provide a useful comparison against the ATM system, especially when the higher bandwidth ATM system did not deliver expected performance levels.

3.4 SPAM/MIDWAY Results

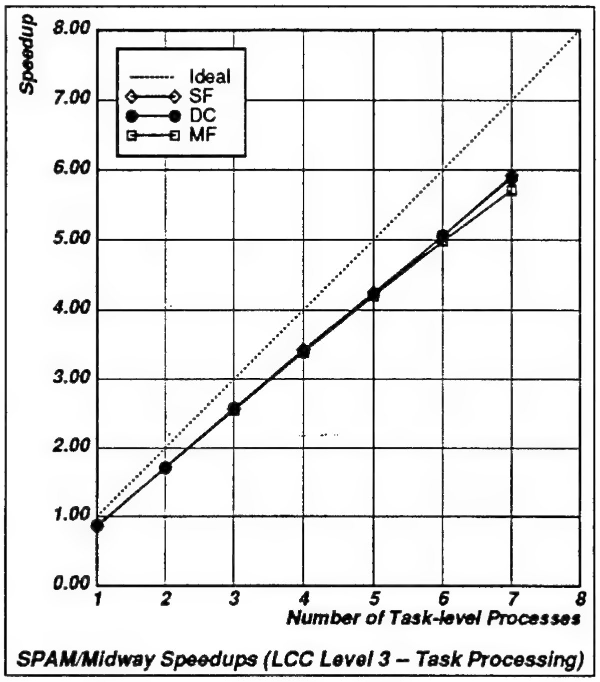
In spite of having to address some of the other performance and stability problems, we have been able to generate improved speed-up results. This implementation of SPAM/MIDWAY has been successfully tested on three airport data sets: Moffett Air Force Base (*mf*), Washington National Airport (*dc*) and San Francisco International Airport (*sf*). The speed-up results are computed against a uniprocessor baseline system (called *faketlp*) that has been modified to include the overheads of building and accessing the task queue.

The hardware for our multi-thread experiments consists of eight DEC 5000/200 machines (each rated at approximately 25 MIPS), each with 64 megabytes of memory. All the machines run MACH 3.0 and are linked via fiber-optic cable using a Fore Systems ATM switch.

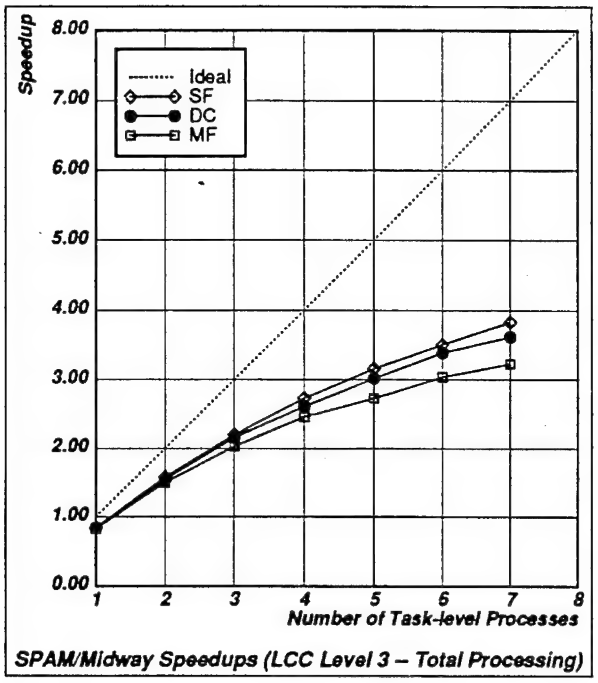
Our current speed-up results for all three data sets using task-level parallelism under MIDWAY are summarized in Figure 5. Table 1 presents the numbers for one of the data sets (*mf*) in more detail. Each row of the table represents a separate system execution. The column labeled **Number of Threads** is the number of child processes used for that run. The **Task Time** column presents the amount of time the system spent executing tasks. This represents the period where actual parallel processing occurs. This begins after the task queue has been set up, and ends before the results are collected together. The **Total Time** column includes task processing time plus the initialization and result collection times.

Number of Threads	**Task Time (sec)**	**Speed-Up**	**Total Time (sec)**	**Speed-Up**
faketlp	835.5	—	880.7	—
1	985.8	0.85	1071.5	0.82
2	491.1	1.70	576.9	1.53
3	330.0	2.53	421.3	2.09
4	248.9	3.36	346.5	2.54
5	200.0	4.18	292.3	3.01
6	168.1	4.97	260.9	3.38
7	145.6	5.74	241.1	3.65

Table 1: Speed-ups varying the number of task-level processes using the MIDWAY virtual shared-memory system for the *mf* data set.



(a) Task Times



(b) Total Times

Figure 5: Speed-ups varying the number of task-level processes using the MIDWAY virtual shared-memory system with the *mf*, *dc*, and *sf* data sets.

We can make several observations from this data. First, the task speed-ups for this system are reasonably good (5.7-fold using 7 child threads) even though we have not tuned the system. The graph in Figure 5 shows that our speed-ups, though below where we would like them to be, again look very linear. If we look at the Task Time data, we can see that the overheads associated with MIDWAY in our system are about 15% per processor. The designers report that, in general, the overheads associated with using MIDWAY are about 10–20% depending on the amount of communication between threads. These numbers compare favorably.

However, when significant amounts of communication occur (such as during the initialization and return-results portions, measured as part of the Total Time), the overheads seem somewhat high and are a current topic of investigation. As with any parallel system, overheads in the serial processing components of the system are performance bottlenecks and begin to stick out like a sore thumb once performance is brought in line in the parallel components.

Although the speed-ups associated with the overall runtime seem low, it should be noted that both the initialization and result collection portions of the execution are currently serialized. These can be parallelized (they were in versions of our network-shared memory system) to improve the overall runtime.

4 Conclusions and Lessons Learned

Over the final year of this contract, June 1994 - June 1995, we have worked to improve the performance and the robustness of the MIDWAY distributed memory system, as well as to complete the implementation of SPAM on MIDWAY. There remain several unexplored research questions, the most important of which we list below:

- End-to-end SPAM — We completed the implementation of three of the four SPAM phases under MIDWAY. The final phase of SPAM, *model-generation*, was never implemented under MIDWAY due to its relative computational importance. There are two aspects to this. First, our early measurements showed that the time to execute the model-generation phase was less than 1% of the SPAM's total execution time. Second, SPAM's model-generation algorithm is implemented using many fine-grained tasks, and we've already discussed some of the problems that this generates in systems like MIDWAY. Though it would be relatively easy to include this phase as a serial post-processing step, the research priority was such that we felt that there were more important issues, primarily performance related, to address.
- "Pipelined" Parallel SPAM — It was our hope that we would be able to spend a significant portion of our time on a new parallel implementation of SPAM that would allow tasks to be pipelined. That is, tasks from different SPAM phases would be permitted to execute simultaneously. For example, as tasks were generated and dispatched in LCC, tasks could begin execution in FA. This complicates the implementation because it violates many of the assumptions made within the original SPAM design. It's interesting, however, because task domain knowledge can be used to efficiently divide and direct the parallel computation. We have many ideas about how such a system could be efficiently implemented, and we would like to do further investigations in this area if time and resources had permitted.
- Parallel SPAM in PPL — Our influence in the design of the Parallel Production Language (PPL)[Acharya, 1992] and SPAM's subsequent re-implementation in PPL has already produced factor of two speedups over previous serial versions of SPAM. However, PPL's explicit parallelism model will potentially simplify the integration of task-level

parallelism into the current and future versions of SPAM. PPL has already been engineered to simplify a parallel implementation on distributed memory software systems like MIDWAYand PVM. We have already done part of the work to implement a version of LCC on this system but we have not been able to complete this investigation, both in terms of implementation and analysis.

- SPAM vs. scientific applications — We see the need for some additional guidance from distributed shared memory (DSM) designers and perhaps some tools also to help the user in evaluating how his application might map to the DSM system and what the issues are in terms of granularity, data structure design, and data structure access by the underlying algorithms in his system. As we saw with the FA phase, this can be very difficult to appreciate by those who are deeply involved in both sides, the DSM design and the application design, let alone the inexperienced user. There don't appear to be any good or standard ways to express some of these issues and the DSM designers seem at a loss as well to communicate this in any effective way. Statistics such as total lock accesses per second can be misleading, for example, since all locks are not created equal. They have their own data dependent accessing patterns and as a result statistics coming out of the DSM system are often not very helpful. In order to move forward in this area one needs information that describe the dynamics of a particular lock and how that compares to the behavior of other memory accesses. Software tools for extracting and presenting that information are not currently available and are complex to design and implement.

- Robustness of DSM systems — Without a broader range of applications to serve to benchmark the DSM system, diverse and dynamic applications like SPAM will not be adequately served by DSM designs and will be reluctant to invest effort in using a DSM system.

We have shown good speed-ups with our task-level parallelism work on MIDWAY, and have continued toward our goal of having an end-to-end parallel SPAM system that can be used for vision research. Our interaction with the MIDWAY group continues to improve the robustness and usability of the MIDWAY system.

During the course of this three year research contract we have presented research results in the areas of task-level parallelism and language support for large knowledge-based systems. SPAM, our knowledge-based interpretation system, continues to be the vehicle for much of this research. Effectively addressing the issue of scalability is important as we seek to both improve SPAM's performance in it's current task domains (airport and suburban housing scenes) and expand SPAM's task domain.

5 Acknowledgements

Many individuals have contributed to the progress of the research presented in this report. We wish to thank Dr. Abraham Waxman, our COTR, for many helpful discussions concerning basic research directions and technology transfer. We also acknowledge the support of Dr. Oscar Firschein, Advanced Research Projects Agency/SISTO who co-funded this research effort.

Wilson Harvey and Dirk Kalp contributed technical material to this report and largely provided the underlying software developments. From the MIDWAY group, Matt Zekauskas has continued to provide hands-on help and guidance while dealing with a persistent set of performance issues.

Finally, we are also grateful for the help from our colleagues in the Digital Mapping Laboratory, without whom our ardent football discussions would have been decidedly one-sided.

References

[Acharya and Kalp, 1990] Anurag Acharya and Dirk Kalp. Release notes for CParaOPS5 5.3 and ParaOPS5 4.4. This is distributed with the CParaOPS5 release, 1990.

[Acharya, 1992] A. Acharya. PPL: An explicitly parallel production language for large scale parallelism. In *Proceedings of the IEEE conference on Tools for AI*, pages 473–474, 1992.

[Bershad and Zekauskas, 1991] B. Bershad and M. Zekauskas. Midway: Shared memory parallel programming with entry consistency for distributed memory multiprocessors. Technical Report CMU–CS–91–170, Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, April 1991.

[Bershad *et al.*, 1993] B. Bershad, M. Zekauskas, and W. Sawdon. The midway distributed shared memory system. Technical Report CMU–CS–93–119, Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, March 1993.

[Forin *et al.*, 1989] A. Forin, J. Barrera, and R. Sanzi. The shared memory server. In *Proceedings of USENIX — Winter 89*, pages 229–243, January 1989.

[Geist *et al.*, 1993] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam. *PVM3 Users's Guide and Reference Manual.* Oak Ridge National Laboratory, May 1993.

[Gupta, 1991] U. Gupta. *Validating and Verifying Knowledge-Based Systems.* IEEE Computer Society Press, 1991.

[Harvey *et al.*, 1989] W. Harvey, D. Kalp, M. Tambe, D. McKeown, and A. Newell. Measuring the effectiveness of task-level parallelism for high- level vision. In *Proceedings of the DARPA Image Understanding Workshop*, pages 916–933. Morgan Kaufmann, May 1989.

[Harvey *et al.*, 1990] W. Harvey, D. Kalp, M. Tambe, D. McKeown, and A. Newell. The effectiveness of task-level parallelism for high-level vision. In *Proceedings of the 2nd ACM/SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 156–167, March 1990.

[Huertas *et al.*, 1989] A. Huertas, W. Cole, and R. Nevatia. Using generic knowledge in analysis of aerial scenes: A case study. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1642–1648. Morgan Kaufmann Publishers, Inc., August 1989.

[Hwang, 1984] Shang-Shouq Vincent Hwang. *Evidence Accumulation for Spatial Reasoning in Aerial Image Understanding.* PhD thesis, University of Maryland, 1984.

[Kalp *et al.*, 1988] D. Kalp, M. Tambe, A. Gupta, C. Forgy, A. Newell, A. Acharya, B. Milnes, and Swedlow K. Parallel OPS5 user's manual. Technical Report CMU-CS-88-187, Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, November 1988.

[Li, 1988] K. Li. IVY: A shared virtual memory system for parallel computing. In *Proceedings of International Conference On Parallel Processing*, pages 94–101, 1988.

[Matsuyama, 1980] T. Matsuyama. A structural analysis of complex aerial photographs. Technical report, Department of Electrical Engineering, Kyoto University, Kyoto, Japan, April 1980. Ph.D Thesis.

[McKeown *et al.*, 1985] D. M. McKeown, W. A. Harvey, and J. McDermott. Rule based interpretation of aerial imagery. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7(5):570–585, September 1985.

[McKeown *et al.*, 1989] D. M. McKeown, W. A. Harvey, and L. Wixson. Automating knowledge acquisition for aerial image interpretation. *Computer Vision, Graphics, and Image Processing*, 46(1):37–81, April 1989.

[McKeown *et al.*, 1994] David M. McKeown, Jr., Steven Douglas Cochran, Stephen J. Gifford, Wilson A. Harvey, J. Chris McGlone, Michael F. Polis, Stephen J. Ford, and Jefferey A. Shufelt. Research in automated analysis of remotely sensed imagery: 1993–1994.

In *Proceedings of the ARPA Image Understanding Workshop*, pages 99–132, Monterey, California, November 13–16 1994. Advanced Research Projects Agency, Morgan Kaufmann Publishers, Inc. Also available as Technical Report CMU–CS–94–197, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

[Strat and Smith, 1988] T. Strat and G. Smith. Core knowledge systems: Storage and retrieval of inconsistent information. In *Proceedings of the DARPA Image Understanding Workshop*, pages 660–665. Morgan Kaufmann Publishers, Inc., April 1988.

[Zekauskas *et al.*, 1994] M. Zekauskas, W. Sawdon, and B. Bershad. Software write detection for a distributed shared memory. In *Proceedings of the First Symposium on Operating Systems Design and Implementation OSDI.* Morgan Kaufmann Publishers, Inc., October 1994.